Numerical Flow Visualization in a Functional Style

Jeff P.M. Hultquist

Report RNR-89-008

NAS Systems Division
NASA Ames Research Center
Mail Stop 258-5
Moffett Field, CA 94035

June 12, 1989

Abstract

Most visualization tools support the creation of polygonal models which represent a portion of the raw data. These tools support only a limited set of *ad hoc* models, and the creation of new models requires the writing of additional software. We suggest a new approach which treats all model types as functional mappings of data points. This approach simplifies the implementation of traditional models and encourages the development of new models.

We adopt a method loosely based on functional programming. The numerical data is mapped from one form to another, and the mapping functions themselves can be operated upon by "meta-operators" to produce new functions. This functional style can be carried further into the realm of mapping data values onto corresponding geometric representations, which we call "symbols."

We describe an interactive environment, currently under development, which uses a functional paradigm and discrete symbols to display fluid flow data. Example images made with this environment are shown.

1 BACKGROUND

Flow visualization has its roots in the methods used to trace the flows in wind tunnels (amply illustrated in [Van Dyke 1982]). As numerical simulations have become more accurate, a new set of *numerical* flow visualization techniques have been developed. We examine the state of this art.

1.1 Fluid flow simulation

The phenomenon to be studied exists in the "physical" space of everyday life, represented by a grid of sample points. Whereas physical space can be measured in meters along some coordinate axes, the "computational space" is indexed by the integer address of each array location. These points are topologically connected as an orthogonal mesh, but their placement generally follows the curved contours of a solid surface, usually an airfoil. Sample points are often arranged more tightly in regions of computational sensitivity.

The partial differential equations which govern the behavior of fluids are solved using finite-difference techniques over the grid. These computations are performed until the entire grid has converged to a stable set of values. This "flow solution" records the state of a simulated fluid flow as a set of sample points on the curvilinear computational grid. For steady (time-invariant) compressible flows, each grid point records its physical location and the local measures of momentum, density, and energy.

The data points in the grid, together with some interpolation function, define a set of continuous fields. Algebraic and differential functions of these fields define dependent fields of scalar, vector, and general tensor order, defined over the same spatial (and perhaps temporal) domain.

1.2 The visualization process

This numerical data must be presented to a human observer. The most faithful representation is a listing of all the numerical values in the grid, each printed to its full significance. If, for example, the data of interest consisted solely of the total pressure on the wing, such a format would be appropriate. For more data, a graphical format is required.

A common approach uses sets of polygons in object space. These "abstract visualization objects" [Haber 1988] or "models" are visible representations of the simulation data. Each point on the model has a number of surface characteristics, such as hue, lightness, texture, and transparency. These parameters can be used to represent additional values in the data.

"Volumetric rendering" is an attempt to avoid the intermediate step of model construction. Instead, a single scalar field is rendered as a mass of varying translucency, hue, or intensity. Such images can provide a better grasp of scalar field data, but cannot represent fields of higher order.

1.3 Current visualization tools

At NASA-Ames, the current arsenal of analysis and presentation tools provide a Cartesian product of supported geometric models and flow quantities. The user selects the model type to be used and the field to be displayed, and the software produces either an image or a model to be rendered using other software.

One of these tools, PLOT3d [Buning 1985], can construct contour surfaces, line plots, vector plots, and several other models. Any one of about fifty different scalar fields or six vector functions can be depicted. PLOT3d can also integrate "particle paths" through either the velocity or vorticity fields.

Particle paths are very useful but rather time-consuming to compute, so a new tool was written for this application. RIP [Rogers 1987] allows the user to interactively select starting locations for paths on an IRIS workstation. These locations are the seed points for the path integration, which is computed remotely on a Cray supercomputer. This tight linkage of compute power and graphics power allows interactive exploration of the velocity field.

Many useful images have been made with these tools, but their expressive power is limited. They support a fixed set of models over a fixed set of fields and nothing more. The need for more flexible tools is attested to by a proliferation of privately modified versions of PLOT3d.

1.4 New requirements

The goal of this work is to create an interactive environment that enables users to create a wide variety of visualization models.

It should be possible to add new features easily, without the need to rebuild the system from the operating system level.

The system must create color raster images quickly and easily. It must also support the creation of black and white line plots, since this is still the dominant mode of graphics published in scientific journals.

The system should encourage the development of methods for the depiction of several field quantities in a single image.

2 SYMBOLS

The use of discrete symbols as the primitive visual element allows us to support a wide range of visualization techniques. They offer a number of advantages over more traditional models. Symbols can be combined as building blocks to create a wide variety of useful visualization models.

2.1 Definition and related work

We will define a *symbol* to be a graphical representation of numerical data. The position and features of each symbol are the functional result of the underlying data values. Symbols are typically small enough so that several dozen can be displayed in a single image. A collection of symbols can be viewed together to obtain an understanding of the structure of the data.

An early use of symbols is described by Chernoff [1973], who presented collections of human caricatures arranged in two-dimensional scatterplots. Each face represented a k-dimensional sample, with two dimensions encoded in the position of the symbol on the page, and others encoded by the size of the eyes, the length of the nose, and so forth. Detecting trends within the data is transformed into looking for similarities among a crowd of faces.

Similar work has been done by Ellson and Cox [1988] to display the results of a finite-element simulation of an injection molding process. In this animation, the length, hue, and shading of a collection of arrowheads is used to depict the velocity and temperature of the molten plastic.

The VIEW system being developed at UNC-Chapel Hill depicts protein molecules by mapping measures such as atomic radius and charge onto the visual parameters of various geometric primitives.

The efforts of these researchers have demonstrated that multi-dimensional data can be efficiently displayed using discrete symbols which encode information in the position and characteristics of each symbol.

2.2 Selectivity and quantity

In *The Semiology of Graphics*, Jacques Bertin [1983] explored the use of symbols to convey information in two-dimensional graphs. He observes that a symbol has a position on the page and a number of "retinal variables," such as color, size, and shape. His examples show discrete data points plotted as symbols on two-dimensional graphs, with other data represented via the retinal variables.

Each variable, be it planar or retinal, carries with it a set of attributes or "levels." One level is "selectivity," which describes whether an observer can disregard all but a subset of the symbols along that dimension. Bertin claims, and demonstrates with examples, that symbols of differing shape tend to blend together, while (for normally-sighted observers) symbols of different hue can be regarded in distinct groups.

A test for selectivity is based on the the colored figure tests used to detect color blindness. In such tests, a field of randomly sized dots is shown to a subject. Each dot has a color and the observer is asked to identify a figure defined by dots of a uniform color. Since color is a selective variable, most people can see the figure embedded in the dots. We can test the other retinal variables in a similar manner. Bertin asserts that hue, lightness, and orientation are selective; saturation, shape, and size are not.

Another of Bertin's levels is that of "quantity," that is, whether different values of that carrier have some intuitive ordering. Bertin claims that that size, saturation, and lightness are quantitative, while hue, shape, and orientation are not.

The appropriateness of each feature mapping should be judged with respect to the inherent qualities of each feature. Scalar data values should be encoded by quantitative carriers, whereas a selective carrier can be used to depict distinct values or levels within a range of options.

2.3 The representation of depth

High performance workstations can depict three-dimensional structure through the use of object animation. But for more complex objects, which cannot be rendered in real time, and for images which must appear in print, other methods for the depiction of depth must be used.

Of course, stereoscopic methods have been used to depict three-dimensional data; chemistry journals often contain such stereo pair images. In this application, discrete symbols have an advantage over other models, as they have more details to help the viewer to fuse the stereo pairs.

Object depth can be conveyed by a quantitative carrier on each symbol. Such "local" depth cues can be useful but they carry few bits of information; therefore, the depth of any particular symbol may be difficult to judge. Such carriers might be better used for the depiction of flow data.

Since many aerodynamic simulations assume symmetric physical behavior, we can draw the symbols twice, mirrored about the plane of symmetry. The "second view," provided by the reflected symbols, can remove the ambiguity of a flat image.

As an alternative method, the shadows cast by an object can help the observer to make sense of a scene. The position of the symbol in object space is mapped into the planar variable of position on the lower surface. Since discrete symbols cast discrete shadows, the shadows of a group of symbols carry much more information than does the single large shadow of a traditional model.

2.4 Other advantages of using symbols

Symbols can be designed with numerous carriers, including color, size, shape, and orientation. Different symbol types can be combined in a single image to reveal the interaction of different fields.

The collective appearance of symbols can be used to carry information. For instance, the temperature of a fluid can be conveyed by the relative distance between neighboring symbols.

Symbols can be used in concert to construct the traditional models. These larger models, such as contour surfaces, cover most of the image space and only the nearest surface is visible. Translucent rendering of each surface can help, but then each surface becomes harder to visually interpret [Wu 1988]. Symbols can be placed with intervening gaps, which allow symbols throughout the volume to be seen.

The use of symbols for the visualization of data adapts well to a functional programming style. Numerical simulation data can be mapped (possibly in parallel) onto geometric counterparts, under mapping functions which can themselves be modified.

2.5 Design issues

When designing images with symbols, we must consider three separate questions. We will consider each, in turn, in the remaining sections of this paper.

Our data defines a continuous field, so we must identify the locations within that domain at which sample values will be depicted. The work mentioned earlier did not have to consider this problem, even the injection molding depiction used a symbol for *every* finite element. The large number of data points used in aerodynamic simulations precludes such an approach.

We must consider how each sample point will be depicted. This requires us to identify which field values are of interest to the researchers, and how to map each of these values to non-conflicting "graphical degrees of freedom."

Finally, we need to consider how different visualizations can be created within the framework of a single interactive environment. The "input language" supported by the user interface, via keystrokes and mouse operations, must be natural and expressive.

3 MAPPING OF DATA SAMPLES

The raw data from a numerical simulation is overwhelmingly large. A subset of the original data must be identified to serve as a representation of the entire dataset. We can view the original data as a set of arrays, each defining a single field. Mapping operations can take any subset of these fields and produce new data of the same or lower dimension. Such mappings can be composed to produce such representative data.

3.1 Creating new fields

The simplest class of mapping includes those which apply some algebraic function to known quantities at each data point. For example, we can divide the momentum by the density to derive the velocity of the fluid at each data point. The velocity can then be reduced to its magnitude to produce a new scalar field.

Rather than using strictly local functions, we can use a template to compute finite difference approximations of differential fields. In such a fashion, we can approximate the gradient of the pressure or the curl of the velocity.

3.2 Filtering data points

Some of the mapping methods simply select a subset of the original data. An example of this would be the creation of a "slice" out of a three-dimensional dataset, giving us a new two-dimensional grid of flow data. Such a filter can also allow us to select the data points on the surface of the airfoil.

We can test each grid point against some predicate, creating a Boolean array of the same dimension as the input. This "mask" can be used in further function applications to suppress the processing of portions of the data.

3.3 Generating new points

Rather than operating with the sample points originally recorded in the data, we can instead create new points which are interpolated between the points in the original data. The most familiar of such functions is the integration of a path through some vector field from an initial seed point. If that field is the velocity, the resulting list of points is a particle path.

Another example of interpolated data is a collection of points at some iso-valued level in a scalar field. A polygonal tiling of a grid of such points yields the familiar contour surface model.

4 MAPPING TO SYMBOLS

Once the representative points have been selected, they must be converted into a graphical construct for presentation to the user. We will again adopt a style of function application over sets of data, but now the result of each mapping is a set of graphical primitives.

4.1 Spatial mapping

Each geometric primitive has a number of vertices in object space. Each vertex must be defined as the result of some function applied to the data. The function may be strictly local and map a data point onto a symbol solely determined by the values at that point. Alternatively, a template may be used, as in the case of a filter that converts a sequence of points to a sequence of linking line segments.

Most visualizations are constructed by mapping the spatial dimensions of physical space directly onto the spatial dimensions of object space. If the data are also time-variant, this fourth dimension is typically mapped onto the elapsing time of an animated sequence. (Such an approach can mimic empirical flow visualization methods.)

The mapping of dimensions need not follow this pattern. A surface plot of a function of two variables maps the dependent function value into the third spatial dimension of object space. A vector plot is the result of mapping each data point to a line segment, with one endpoint being a function of the data position, and the other endpoint one of both position and velocity.

4.2 Feature mapping

Each geometric primitive is associated with a number of surface characteristics, including color, shininess, and transluscency. These additional degrees of freedom can be determined as the functional result of values in the data. The use of local and template functions can be applied just as in spatial mappings with the result applied to any characteristic supported by the rendering and output systems.

For convenience, the color of new primitives should probably be set to some user-controlled default value. The color of any given primitive could then be allowed to take the default value, specified explicitly at the time the primitive is created, or altered by mapping a set of primitives through yet another filter.

Note that these characteristics do not carry as much information as do the spatial variables. Information which is clearly visible on a display may be obscured in a hard-copy version of the same image. Furthermore, since some people are color blind, it is unwise to depend on color for the *presentation* of results.

5 AN EXAMPLE IMPLEMENTATION

An environment is being built at NASA-Ames which supports the visualization of fluid flow data using a functional approach. It is called *PROVE*, an acronym for "Programmable Visualization Environment." This environment has been designed to be as extensible as possible, so that new visualization techniques can be created within its framework. The system is expected to evolve in response to the needs of the user community.

5.1 Overview

The system is being built with the language Scheme, a concise dialect of LISP. The interactive nature of Scheme allows us to add new features to a running environment, and to load entire modules on demand. Scheme automatically garbage collects data which is no longer in use, relieving the programmer from the task of managing free storage.

PROVE runs on Silicon Graphics IRIS-4 workstations, under ForeSight, the standard window manager. In its initial configuration, the system presents a text window running the GNU Emacs text editor. The Scheme interpreter runs as a subprocess under the editor and has been extended to have access to most of the functions in the IRIS graphics library. Everything that appears on the screen is placed there by the Scheme interpreter.

The environment contains a number of modules which support management of simulation data, vector algebra, mapping functions, and an ever-increasing set of features. At present, almost all user interaction with the system is textual. As useful features are developed, they will be made available as menu options. We hope that over the next few years a large collection of graphics tools will be developed to run under *PROVE*.

In Scheme, the "list" is the most common data structure. The function `map` can produce a new list by applying a function over the corresponding elements of a number of lists of equal length. For example, the expression

```
(map + '(10 12 14) '(1 2 3))
```

evaluates to the new list `'(11 14 17)`. In *PROVE*, we have implemented a new data structure called a "grid" which represents a n-dimensional array of values. The function `map-grid` takes a function, an input template, and a number of isomorphic grids. It produces a new grid which contains the results of the function and template applied at each base location in the original grids. (Output values for which the input template exceeds the grid boundaries are marked as "undefined," and the result of any function which has undefined inputs is also "undefined.")

In the remainder of this section, we present a number of images created with *PROVE*. The data are from a simulation of flow over a delta wing. They were provided by John Ekaterinaris, a researcher at NASA-Ames.

5.2 Density plot

In the first example, we construct a line drawing which depicts the fluid density contours on the upper surface of the airfoil (plate 1). This is a simple example meant to illustrate the use of functional mappings in the generation of an image. We also wish to emphasize that scientific journals rarely contain anything but line art such as this, so visualization tools should be able to produce such output.

We first select the data points across the upper surface. This new two-dimensional grid is examined to find the extreme values, and this range is subdivided into a number of levels. Each cell, defined by an input template, is processed by a contouring filter to create a set of line segments. The edges of each cell are also converted to line segments to produce the underlying grid lines. All of these segments are mapped to the image plane and saved as a PostScript file. The numerical labels along the right-hand edge are placed by running a contour filter on that edge.

5.3 Ribbons

A common model is created by integrating a set of starting points to create particle paths through the velocity field. We select a set of points along the leading edge of the airfoil, and integrate these through the velocity fields. This produces a set of one-dimensional arrays of data points. We can map each pair of adjacent points to a line segment and produce the traditional particle path visualization model (plate 2).

If we select pairs of paths and tile the gap between them, we create ribbons which twist downwind from the airfoil. (Such ribbons were used by Belie [1985] to depict flow in the main engines of the space shuttle.) We can assign color to these polygons, alternating the color as we proceed down the list. This yields a "dashed" ribbon (plate 3), which conveys information about the velocity and the divergence of the flow.

5.4 Combination image

We have seen how an image of particle paths can be made more useful by mapping the sample points onto sequences of polygons. This approach is used in the design of a final image which depicts a number of quantities.

We first select all those points at which the density has a value less than 92% of the standard value. Those points which survive this winnowing are mapped onto red points in the object space. Grid points at which the velocity has a component back toward the nose of the airfoil are marked, and those that can be matched with orthogonal neighbors become the endpoints of black line segments. Finally, slices orthogonal to the airfoil are selected. Each slice is processed by the contouring filter from the first example, this time to create contour lines at which the "stagnation energy per unit volume" has a value of

1.7. These contour lines are added to the growing collection of primitives to be rendered. Plate 4 shows all of these models, mapped into object space and rendered with drop shadows.

6 CONCLUSION

The construction of models in object space is a popular method of visualizing data. These models are often constructed by software specific to each model. By adopting the methods of functional programming, we are able to create a wide variety of models using sequences of very simple functions.

By emphasizing the creation of discrete symbols as the functional result of collections of data points, we encourage the development of new models. Furthermore, symbols which are placed individually in object space can carry more information than is possible with more traditional models.

The *PROVE* software is a flexible base from which to develop new visualization techniques. We now need to identify the most useful of these and make them widely available to the scientific community.

7 ACKNOWLEDGEMENTS

Mary Hultquist edited this paper and caught many errors. Dave Kerlick and Creon Levitt made suggestions that lead to some of the nicer features of this work. Fred Brooks and Henry Fuchs made the first suggestion that a mathematical structure for visualization was worth investigating. John Ekaterinaris graciously allowed the use of his numerical flow data. Lakshmi Dasari explained how this work relates to real functional programming.

8 BIBLIOGRAPHY

Belie, R.G. [1985] Flow Visualization in the Space Shuttle's Main Engine, *Mechanical Engineering*, (September), 27–33.

Bertin, J. [1983] *The Semiology of Graphics.* (translated by W. Berg), Madison: University of Wisconsin Press.

Buning, P.G. and Steger, J.L. [1985] Graphics and Flow Visualization in Computational Fluid Dynamics. In *Proceedings of the AIAA 7th CFD Conference* (Cincinnati, July). AIAA# 85-1507-CP.

Chernoff, H. [1973], The Use of Faces to Represent Points in k-Dimensional Space Graphically. *Journal Amer. Statistical Assoc.*, 68 (June), 361–368.

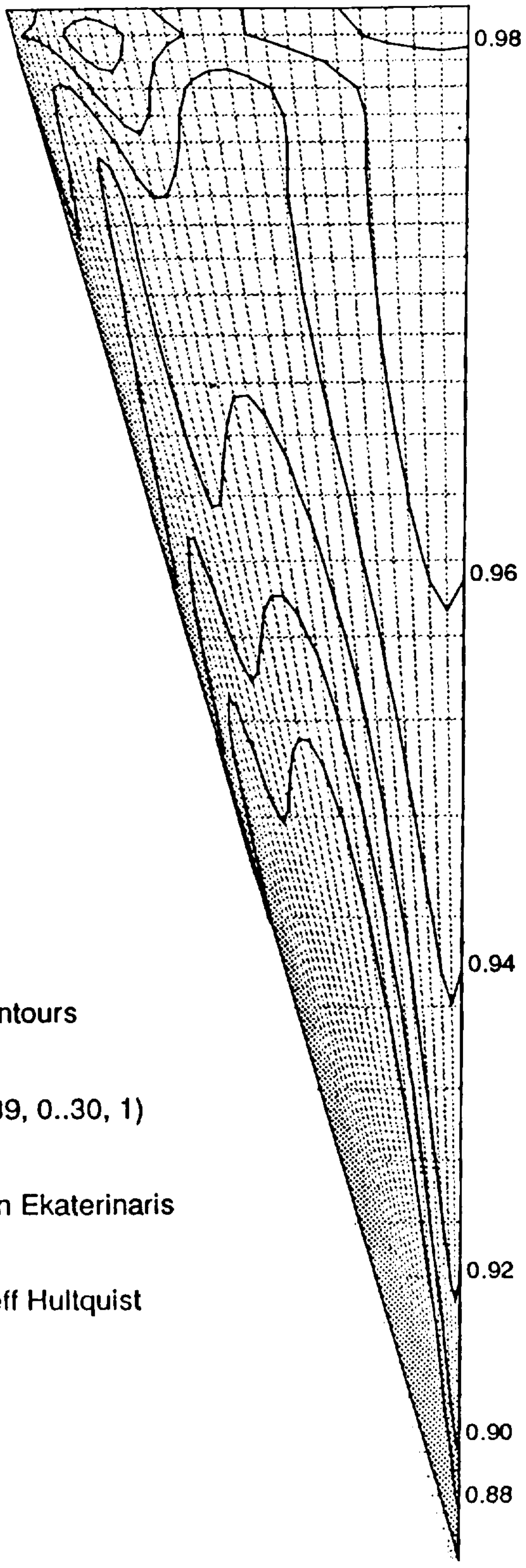
Ellson, R. and Cox, D. [1988] Visualization of Injection Molding. *Simulation.* 51, 5 (Nov. 1988), 184–188.

Haber, R.B. [1988] Visualization in Engineering Mechanics: Techniques, Systems, and Issues. In course notes for *Visualization Techniques in the Physical Sciences (#19)*, SigGraph, (Atlanta, August 1-5).

Rogers, S.E., Buning, P.G., and Merrit, F.J. [1987] Distributed Interactive Graphics Applications in Computational Fluid Dynamics. *International Journal of Supercomputer Applications.* 1, 4 (Winter 1987), 96–105.

Van Dyke, M. [1982] *An Album of Fluid Motion.* Stanford: The Parabolic Press.

Wu, K. and Hesselink, L. [1988] Computer display of reconstructed 3-d scalar data. *Applied Optics.* 27, 2 (15 Jan. 1988), 395–404.



density contours

(ijk) = (0..39, 0..30, 1)

data: John Ekaterinaris

image: Jeff Hultquist

plate 1

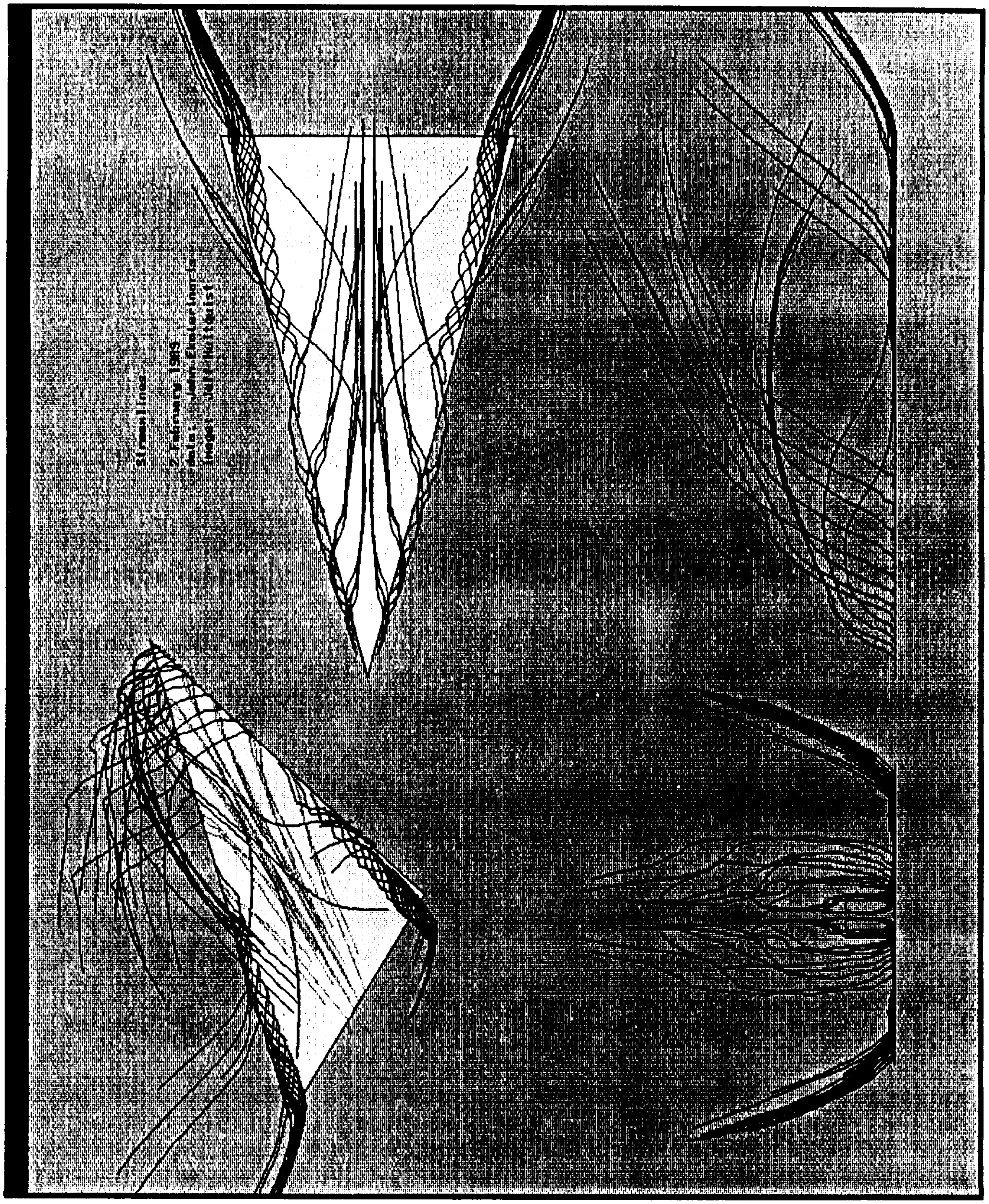

Streamlines
7 February 1989
data: John Ekaterinaris
image: Jeff Hultquist

plate 2

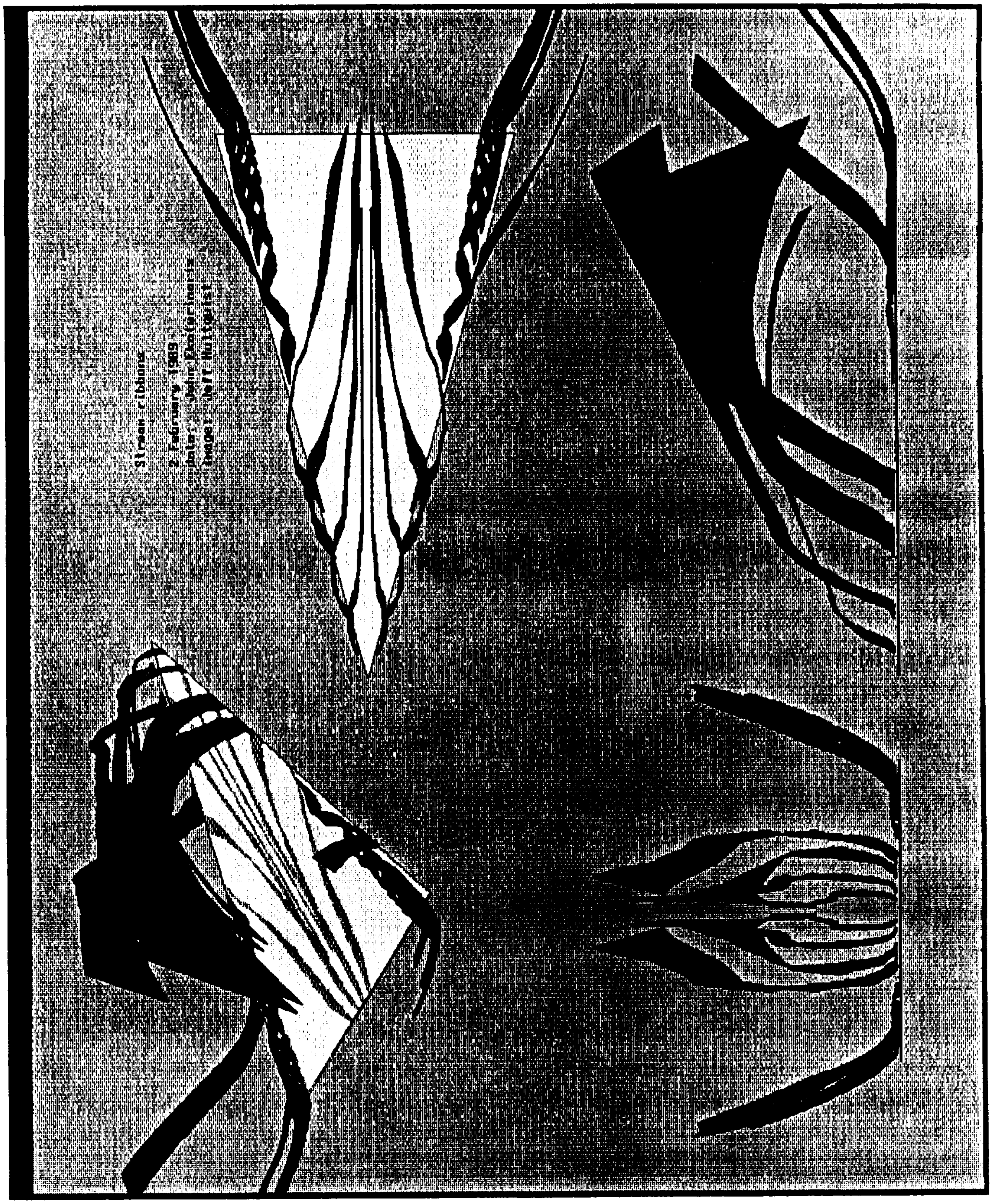

Stream ribbons
2 February 1989
data: John Ekaterinaris
image: Jeff Hultquist

plate 3

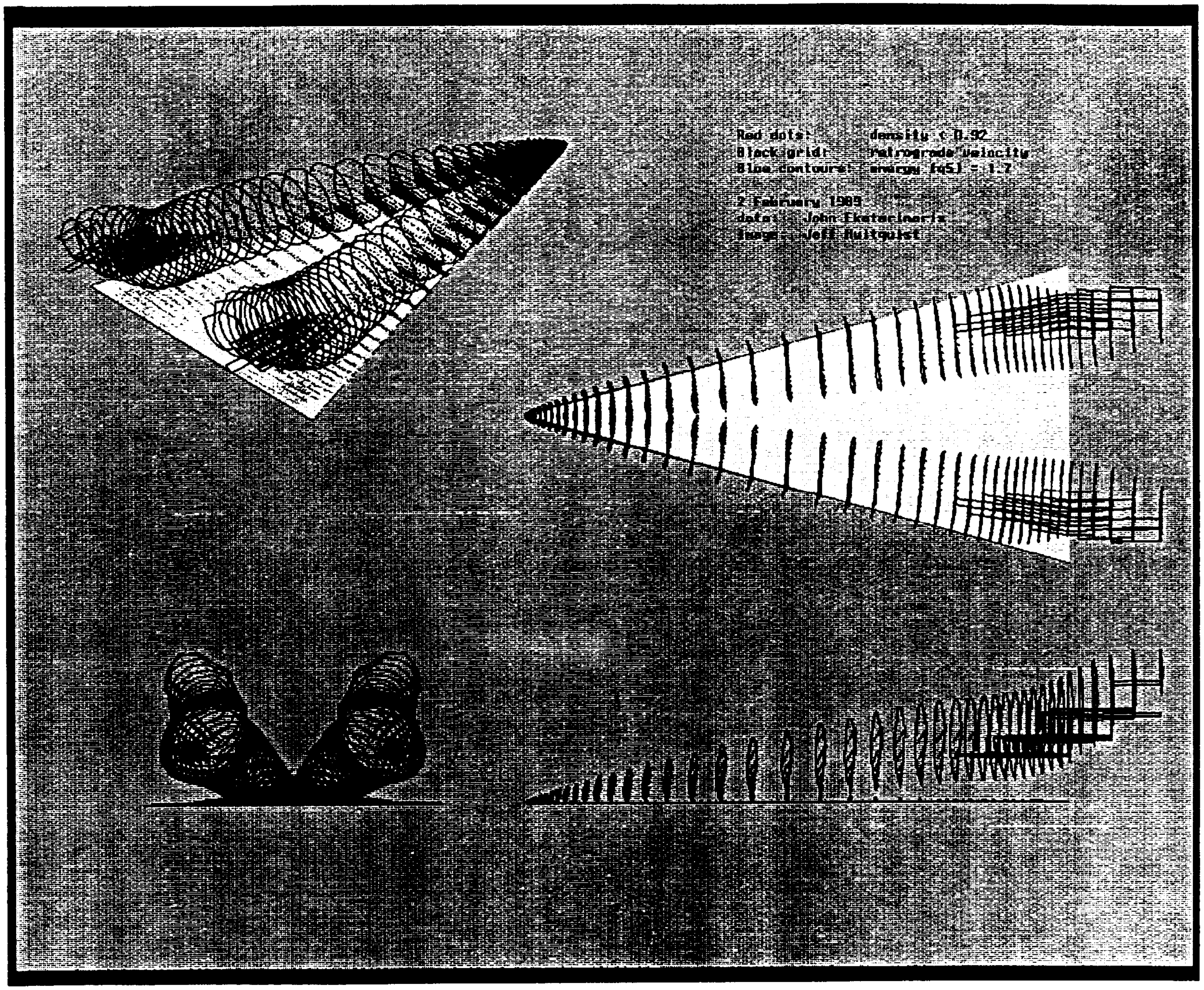

Red dots: density < 0.92
Black grid: retrograde velocity
Blue contours: energy (q5) = 1.7
2 February 1989
data: John Ekaterinaris
Image: Jeff Hultquist

plate 4